VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MOBILNÍ APLIKACE PRO STUDIS VUT

MOBILE APPLICATION FOR STUDIS BUT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **JAN SMYČKA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAKUB ŠPAŇHEL**
SUPERVISOR

BRNO 2018

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2017/2018

Zadání bakalářské práce

Řešitel: **Smyčka Jan**
Obor: Informační technologie
Téma: **Mobilní aplikace pro Studis VUT**
Mobile Application for Studis BUT
Kategorie: Uživatelská rozhraní

Pokyny:

1. Seznamte se s problematikou návrhu a vývoje mobilních aplikací; zaměřte se na platformu Android.
2. Seznamte se s problematikou webových aplikací a tvorbou API pro komunikaci s mobilní aplikací.
3. Analyzujte možnosti získávání dat z informačního systému Studis VUT a navrhněte možná řešení.
4. Navrhněte a prototypujte způsob interakce s aplikací a jednotlivé prvky uživatelského rozhraní.
5. Navrhněte a implementujte řešenou mobilní aplikaci.
6. Testujte vytvořenou aplikaci na uživatelích a iterativně ji vylepšujte.
7. Zhodnoťte dosažené výsledky a navrhněte možnosti pokračování; vytvořte plakátek a krátké video pro prezentování projektu.

Literatura:

- Android Developers: https://developer.android.com/index.html
- Susan M. Weinschenk: 100 věcí, které by měl každý designér vědět o lidech, Computer Press, Brno 2012

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Špaňhel Jakub, Ing.**, UPGM FIT VUT
Konzultant: Marušinec Jaromír, Ing., Ph.D., CVIS VUT
Datum zadání: 1. listopadu 2017
Datum odevzdání: 16. května 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Bakalářská práce se zabývá vývojem mobilní aplikace v operačním systému Android pro informační systém VUT. Teoretická část práce se věnuje operačnímu systému Android a jeho základním prvkům užívaným při tvorbě aplikace, v textu je rozebrán také design a architektonický návrh mobilních aplikací. Zásadní část práce se věnuje popisu implementace vytvořené aplikace a jejímu testování na uživatelích. V závěrečné části práce je vývoj aplikace a testování uživatelů zhodnoceno a je navržen další možný vývoj aplikace do budoucna.

Abstract

Bachelor thesis deals with the development of mobile application in Android for VUT information system. The theoretical part of the thesis describes Android operating system and its basic elements used in the application development process, the text also describes the architectural design of mobile applications. The main part of the thesis deals with the implementation of the application and its testing on the users. In the final part of the thesis, the application development and user testing are assessed, and further possible development of the application for the future is proposed.

Klíčová slova

Android, uživatelské rozhraní, informační systém, komponenty architektury, Model-View-ViewModel, vkládání závislostí, Dagger2, Repository

Keywords

Android, user interface, architecture components, Model-View-ViewModel, dependency injection, Dagger2, Repository

Citace

SMYČKA, Jan. *Mobilní aplikace pro Studis VUT*. Brno, 2018. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jakub Špaňhel

Mobilní aplikace pro Studis VUT

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jakuba Špaňhela. Další informace mi poskytl Ing. Jaromír Marušinec Ph.D., MBA. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jan Smyčka
17. května 2018

Poděkování

Tímto bych chtěl velmi poděkovat vedoucímu práce, Ing. Jakubu Špaňhelovi, za konzultace, cenné rady, které vedli k dokončení této práce a že byl ochoten převzít vedení této práce.

Obsah

Kapitola 1

Úvod

Předkládaná bakalářská práce se zaměřuje na vytvoření mobilní aplikace pro Studis VUT pro operační systém Android. Jedná se o projekt, který začal ve spolupráci s Centrem výpočetních a informačních služeb VUT.

Teoretická část práce se zabývá operačním systémem Android, který vznikl jako start-up v roce 2003 a původně byl navržen jako operační systém pro mobilní fotoaparáty. Po dalším vývoji a zakoupení firmou Google je dnes využíván především v mobilních telefonech, tabletech, televizorech ale také automobilech a patří mezi jednoznačně nejpoužívanější operační systémy. Operační systém Android je veden jako otevřený systém, tzv. open source, což podporuje aktivní vývojáře a vznik nových aplikací.

V práci jsou popsány základy tvorby aplikace pro Android - základním kamenem při tvorbě aplikace pro Android je tzv. Android manifest, tedy XML soubor, který definuje důležité nezbytné informace pro vývoj aplikace. Definuje například povolení, která si aplikace žádá, aby mohla komunikovat s hardwarem zařízení či s jinými aplikacemi. Komunikace (provádění jednotlivých aktivit či služeb) se v OS Android provádí skrze Intenty a filtry, které jednotlivé intenty zachytí a zpracují. V manifestu aplikace jsou také definovány základní prvky uživatelského rozhraní aplikace, v případě této bakalářské práce, jsou jimi aktivita a fragment, jehož využití umožňuje zobrazení více na sobě nezávislých aktivit a tedy větší flexibilitu při zobrazování obsahu aplikace na různých zařízeních.

Část práce je věnována komponentám architektury, které byly firmou Google představeny v roce 2017 a které mají za cíl usnadnit vývoj aplikací a tvorbu robustního kódu. Jedná se o komponenty Lifecycle, Live Data, ViewModel a Room.

Zvláštní kapitola se zabývá designem a architektonickým návrhem aplikace, který je u každé aplikace zásadní pro to, aby byla uživatelsky přívětivá a umožňovala rychlou navigaci skrz aplikaci. Praktická část práce popisuje implementaci aplikace pro Studis. Jelikož má aplikace potenciál pro další vývoj, drží se implementace pravidel pro psaní „čistého kódu“ jako je například princip jedné zodpovědnosti či dostatečná expresivita kódu.

Cílem této práce je také vytvořenou aplikaci otestovat na studentech, zhodnotit jejich zpětnou vazbu a integrovat jejich připomínky. Součástí práce je návrh plakátu a krátké video pro prezentování aplikace a v závěrečné části práce jsou navrženy možnosti pokračování vývoje aplikace.

Kapitola 2

Operační systém Android

2.1 Android

Android je operační systém původně navržen pro digitální fotoaparáty[1]. Vznikl v roce 2003 v rámci startupu[1], klíčovým momentem pro Android jako operační systém pro mobilní zařízení byl ale rok 2005, kdy byl počáteční startup zakoupen firmou Google, která směřovala jeho využití právě na mobilní zařízení. Dnes je Android využíván i v řadě jiných zařízení jako jsou herní konzole, televizory či automobily.

Android je vytvořen na základě upravené verze jádra (kernelu) Linuxu. Vývoj je veden jako **open-source** projekt[2]. V kontrastu s tímto přístupem, druhý hráč na poli mobilních operačních systémů, iOS od Apple, udržuje svůj ekosystém velmi uzavřený a vývoj pro něj je komplikovanější.

Každá verze Androidu je kódově pojmenována podle sladkostí v posloupnosti podle abecedy (*Cupcake*, *Donut*, *Eclair*,...). Toto pojmenování se začalo používat až od Androidu verze 1.5 s písmenem **C**, jako *Cupcake*.

Jednou z nevýhod toho, že operační systém Android je open-source je variabilita implementace systému jako takového. Toto může vést k neočekávanému chování na telefonech méně známých značek.

V dnešní době je zastoupení Androidu na trhu s mobilními zařízeními přes 70 %[3].

2.2 Základy tvorby aplikace

V následující části budou popsány základní elementy, které se používají při vývoji Androidu. Znalosti o vývoji Androidu jsem získal převážně z **The Busy Coder's Guide to Android Development** [3].

2.2.1 Android manifest

Android manifest je XML soubor, který definuje důležité informace o aplikaci – pojmenování, komponenty, povolení, obrazovky.

[1] zakladatelé Rich Miner, Nick Sears, Chris White, a Andy Rubin
[2] Android Open Source Project (AOSP) - https://source.android.com/
[3] StatCounter Global Stats, http://gs.statcounter.com/os-market-share/mobile/worldwide

Povolení

Aplikace mají možnost používat zdrojů zařízení (např. čtení SD karty, připojení na internet, kalendář). Aby měla aplikace přístup k těmto prvkům, je třeba získat **povolení (permission)**.

Jejich získávání se liší podle verze Androidu, pro dřívější verze byla povolení žádána při instalaci aplikace, od Androidu Nougat 6.0 jsou získávána *runtime*[4] – jakmile jsou zdroje potřeba.

Existují čtyři kategorie povolení pro zdroje zařízení:

- **Bezpečné**/*Normální* – jsou považovány za bezpečné z hlediska informací o uživateli (nevyžadují explicitní potvrzení povolení pro verzi Androidu 6.0+),
- **Nebezpečné** – jedná se o použití zdrojů, které by mohly potencionálně kompromitovat uživatelova data,
- **Podepisovací a Speciální** – jedná se o velmi specifické případy a při běžném vývoji aplikací nejsou použity (např. zobrazování oken přes ostatní aplikace – Messenger od Facebooku).

Existují i *skupiny povolení*, touto cestou lze jedním příkazem nadefinovat více než jedno.

Intenty a jejich filtry

Intenty jsou prvkem komunikace. Může se jednat o komunikaci se systémem (startování aktivit, případně **Služeb**[5]) nebo i s ostatními aplikacemi v zařízení skrze **filtry (Intent filters)**. Intent si lze představit jako balíček, který se pošle do systému, kde, je-li na něj filtr přihlášen, tak jej intent-filtr zachytí a zpracuje.

Intenty se dělí na:

- **Implicitní** – nedefinují, která komponenta má intent zpracovat, ale obsahují informace, pomocí kterých systém sám rozeznává, která komponenta ho má zpracovat – touto cestou lze nabídnout použití aplikace na předem nadefinované intenty v Android systému,
- **Explicitní** – mají přímo nadefinovanou komponentu, která si intent má převzít – typickým příkladem je startování aktivit (např. *Intent(StudentMainActivity.class)*).

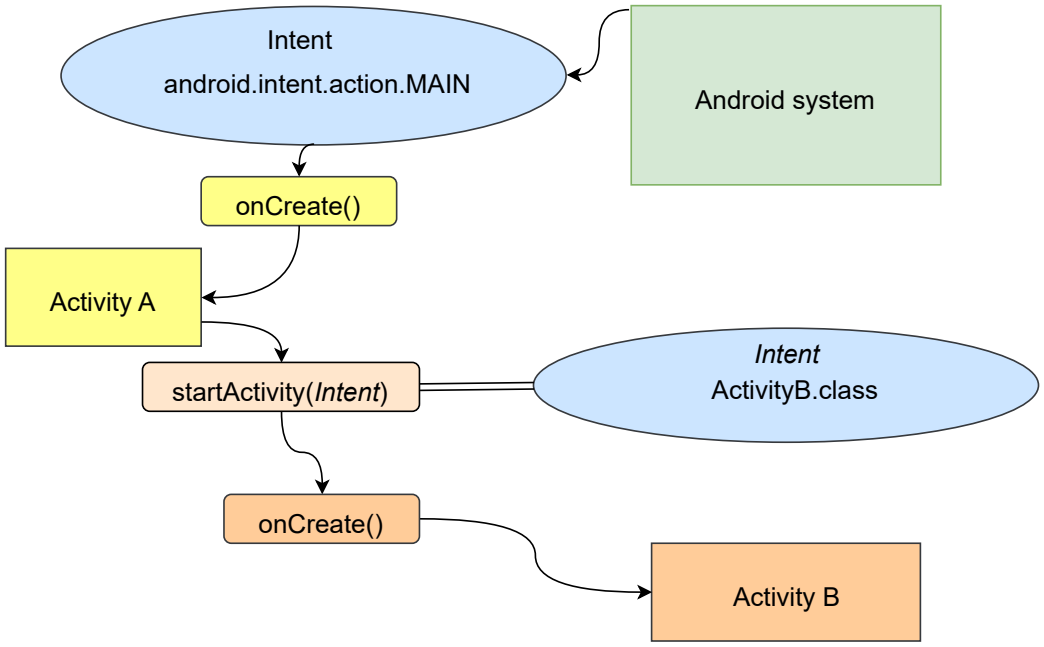
Element, který určuje, co se s intentem stane, je element ***action***. Tímto elementem se také definuje vstupní bod aplikace (výp. 2.1) – aktivita má zaregistrován filtr se specifickým atributem **MAIN** (obr. 2.1).

```
<activity android:name=".LoginScreen">
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
</activity>
```

Výpis 2.1: Příklad aktivity s filtrem definované jako vstupní bod v manifestu

[4] runtime (za běhu) – uživateli je zobrazeno okno s žádostí

[5] služby (Services) – operace, které běží v pozadí – https://developer.android.com/guide/components/services.html



Obrázek 2.1: Zobrazení startu aplikace skrze *MAIN* intent a následný přechod do druhé aktivity pomocí explicitního intentu

Filtry mají dvě možnosti deklarace – **statickou** a **dynamickou**.

Statická deklarace (výp. 2.1) je prováděna skrze manifest, při níž je filtr neustále aktivní a *poslouchá* na možné intenty, což se dá považovat za výhodu, ale zároveň nevýhodu z hlediska náročnosti na zdroje zařízení (např. baterie).

Statická deklarace může vést ke zpomalování systému, zejména, je-li na zařízení velké množství aplikací, které poslouchají na daný intent (všechny tyto aplikace jsou v tomto případě aktivovány a vyhodnocují, zda budou na intent reagovat).

Druhou alternativou deklarace je deklarace dynamická, která je spojená s **contextem** (kap. 2.2.5) aktuálně zobrazované aplikace/prvku rozhraní, tzn. filtr je aktivní tak dlouho, jak dlouho existuje context.

Realný dopad nevýhod statické deklarace

Hlavním intentem, na kterém lze znázornit nevýhody statické deklarace je intent CONNECTIVITY_ACTION – při přecházení z mobilních dat na WiFi mohl být vyslán mnohokrát a každá aplikace si žádala systémové zdroje pro jeho zpracování. Od verze Marshmallow 7.0 je tento síťový intent skrze statickou deklaraci (v manifestu) nedostupný. Tento intent byl používán velkým množstvím aplikací, což vedlo k velkému zpomalování systému.

2.2.2 Životní cyklus prvků uživatelského rozhraní

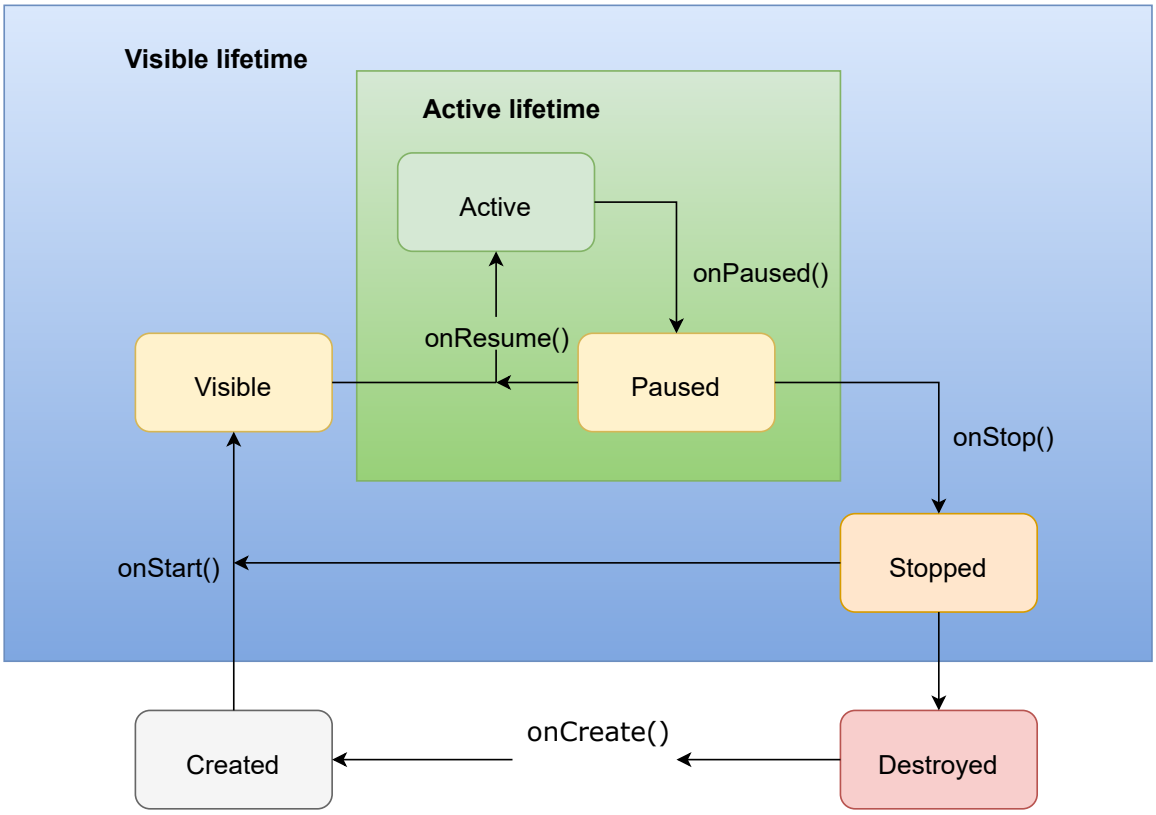
Životním cyklem označujeme sérii stavů. V našem případě se jedná o cyklus prvků uživatelského rozhraní aplikace – prvků **Aktivita** (kap. 2.2.3) a **Fragment** (kap. 2.2.4). Správné získávání a uvolňování zdrojů je důležité pro stabilitu aplikace.

Oba dva prvky jsou řízeny přechody stavů životního cyklu. Každý prvek má svou specifickou variantu cyklu, která je založena na společném principu (obr. 2.2).

Přechody mezi stavy životního cyklu jsou doprovázeny funkcemi, které lze přetížit, což je základem implementace prvků Androidu.

Jedná se o funkce:

- **onCreate** – stará se o vytváření a načtení uživatelského rozhraní, jako jedinné z těchto volání je povinná vlastní implementace, z důvodů nastavení konkrétního rozhraní,
- **onStart** – rozhraní (aktivita/fragment) se stane viditelné uživateli – **Visible**,
- **onResume** – prvek rozhraní se stane aktivním – **Active** (není ničím překrýván),
- **onPause** – prvek rozhraní je překryt, ale některé části jsou stále viditelné uživateli,
- **onStop** – žádný z prvků rozhraní není viditelný,
- **onDestroy** – koncové volání, obecně používáné pro uvolňování zdrojů alokovaných v **onCreate**, existují ale i případy, že není zavoláno – uvolňování je třeba řešit jinak.

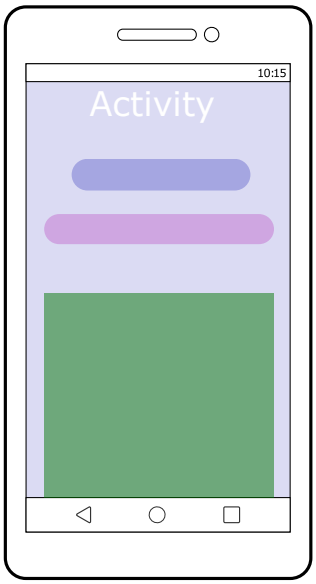


Obrázek 2.2: Zoobecnění životních cyklů dvou hlavních prvků rozhraní (Aktivita a Fragment). Znázornění stavů, kdy je prvek aktivní (Active lifetime) a kdy pouze viditelný uživateli (Visible lifetime).

Prvek rozhraní může projít těmito stavy mnohokrát a je na vývojáři, aby zajistil konsistentní chování aplikace.

2.2.3 Aktivita

Aktivita je základním stavebním blokem pro Android. Obecně by se dalo říct, že reprezentuje obrazovku aplikace (obr. 2.3) a vstupní bod pro uživatele.



Obrázek 2.3: Znázornění, jak lze chápat aktivitu

Aktivita nastavuje a drží si informace o právě zobrazovaném uživatelském rozhraní, které je zpravidla nadefinováno pomocí XML souborů. V případě potřeby dynamického chování je možné přidávat prvky rozhraní i přímo v kódu.

Aplikace není počtem aktivit omezena, ale každou aktivitu je třeba deklarovat v manifestu aplikace (výp. 2.3).

```
<activity
        android:name=".login.LoginActivity"
        android:label="@string/app_name"
        android:noHistory="true"
        android:theme="@style/MerkurAppTheme"
        android:screenOrientation="portrait"
        android:windowSoftInputMode="adjustPan">
</activity>
```

Výpis 2.2: Příklad deklarace aktivity v manifestu

Chování při změně orientace

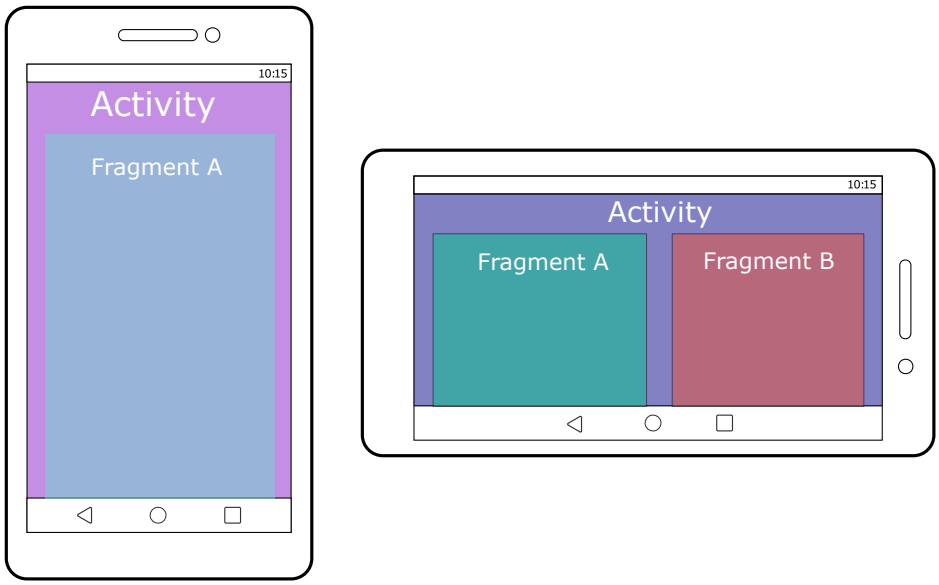
Změní-li uživatel orientaci zařízení, systém uklidí stávající instanci aktivity a vytvoří uživ. rozhraní kompletně znovu, novým cyklem.

Funkce **onCreate** má jeden parametr – *savedInstanceState* – balíček, do kterého lze uložit data, která chceme zachovat. Toto je mechanismus, jak předat nové instanci aktivity informaci o předchozím stavu.

Je čistě na vývojáři, jak uchová jednotlivé hodnoty v rozhraní – používájí-li se komplexní data, není jednoduché cesty, jak je předávat skrze *savedInstanceState*.

2.2.4 Fragment

S růstem trhu mobilních zařízení ve směru k tabletům a větším obrazovkám přestávala být aktivita jediným vhodným prvkem rozhraní. Omezení, že pouze jedna aktivita může být v daný čas aktivní a problém, který toto omezení vytvořilo (aktivity se stávaly *God object*[6]) vedlo k přidání nového prvku rozhraní - s příchodem verze Androidu 3.0, Google představil další prvek rozhraní – **Fragment** (obr. 2.4).



Obrázek 2.4: Znázornění možností využití fragmentů, je-li zařízení orientováno na šířku

Zjednodušeně lze fragment označit ve všech hlavních ohledech za aktivitu. Avšak jeho charakteristiky nejsou spojené s jednotlivou obrazovkou/aktivitou, ale spíše s jeho obsahem. Fragment bývá zpravidla umísťován do kontejnerů v uživ. rozhraní, díky čemuž jsou fragmenty nezávislé na konkrétní aktivitě.

Životní cyklus fragmentu je velmi blízký společnému principu (obr. 2.2).

Google se snaží směrovat vývoj právě k používání fragmentů. Ty byli dlouho považovány za komplikované a nestabilní. Google se poslední léta snaží tyto nedostatky odstranit.

2.2.5 Context

Context jako prvek v Androidu se dá chápat jako informace o aktuálním prostředí/stavu aplikace. Je to rozhraní, které je implementováno několika hlavními komponentami – **Aplikace**, **Aktivita**, Služba, ContentProvider, BroadcastReceiver.

Context se používá pro systémovou interakci a alokaci zdrojů. Právě ono alokování zdrojů může snadno vést k problému s pamětí (případně s baterií) – v případě, že aktivita zaregistruje nějakou službu systému skrze její context a spravně ji neodregistruje. Přestane-li komponenta, jejíž context posloužil k její aktivaci, existovat, služba bude mít stále referenci na komponentu a dojde k úniku paměti.

Používá se proto takový context, který bude existovat tak dlouho jako funkce, kterou s ním chceme provést. Pro přidávání prvků do uživatelského rozhraní avšak musíme použít

[6]God object – objekt, který má přílíš mnoho zoodpovědností a toto vede ke komplikaci při vývoji a zhoršuje čitelnost kódu

contex dané aktivity. Pro vytváření úkonu v nových vláknech je možné/správné použít *context* aplikační z důvodu možné neexistence aktivity při skončení práce ve vlákně.

Tento princip využívá i nedávno představená komponenta **ViewModel** (kap. 2.3.3).

2.2.6 SharedPreferences

SharedPreferences (SP), neboli *sdílená nastavení*, umožňují ukládat jednoduchá data, nejčastěji různá nastavení aplikace. Data jsou uložena do XML souboru ve stylu **klíč–hodnota**.

Pomocí SP lze i vytvářet jednotlivé prvky rozhraní[7], kde se jejich možnosti/hodnoty předem nadefinují v XML souborech. Při použití rozhraní skrze SP není třeba implementovat jednotlivé reakce na vstupy od uživatele, ale systém je zpracuje sám (výp. 2.3).

```
<ListPreference
        android:defaultValue="0"
        android:entries= "@array/faculty_theme_entries"
        android:entryValues="@array/faculty_theme_values"
        android:key="theme_color"
        android:negativeButtonText="@string/cancel"
        android:positiveButtonText="@string/confirm"
        android:summary="@string/theme_color_summary"
        android:title="@string/theme_settings" />
```

Výpis 2.3: Příklad deklarace jedné položky SharedPreferences. Po otevření se uživateli zobrazí nabídka na výběr z možností nadefinovaných v *faculty_theme_entries* a následně se uloží jejich hodnota (korespondující indexu v *faculty_theme_values*)

2.3 Android - komponenty architektury

V roce 2017 Google představil na své každoroční konferenci, Google I/O, prvky architektury, které mají za cíl usnadnit vývoj aplikací a tvorbu robustního kódu – **Architecture components** (Lifecycle, LiveData, ViewModel, Room).

Informace a vědomosti popsané v této kapitole jsem čerpal z e-knih **Android's Architecture Components** [2].

2.3.1 Lifecycle a LifecycleObserver

Za účelem zjednodušit spravování zdrojů a služeb systému byla vytvořena třída **Lifecycle** reprezentující životní cyklus (kap. 2.2.2).

Bylo také vytvořeno rozhraní, pomocí kterého lze na změny životního cyklu reagovat – **LifecycleObserver**.

Třída Lifecycle představuje referenci na cyklus prvků uživatelského rozhraní (např. aktivity), má informace o jeho stavu a umožňuje poslouchat na změny životního cyklu. Toto umožňuje přenechat implementaci potřebných akcí při přechodu mezi stavy cyklu na dané komponentě (výp. 2.4) – komponenta se stane vědoma o cyklu – **Lifecycle aware**.

Používala-li aktivita více komponent, které potřebovaly informace o tom, v jakém je aktivita stavu, musela aktivita řešit předávání těchto informací.

Přesun této zodpovědnosti do komponenty samotné (nemusí se vždy jednat o komponentu Androidu ale tu, kterou jsme vytvořili sami) vedl k vyšší míře čitelnosti kódu a

[7] např. PreferenceFragment – https://developer.android.com/reference/android/support/v7/preference/PreferenceFragmentCompat

modularitě komponent (ve chvíli, kdy končil cyklus rodičovského prvku, komponenta má možnost vykonat všechna potřebná uvolnění zdrojů sama).

```
public class MyComponent implements LifecycleObserver {

        MyComponent(Lifecycle lifecycle) {
                // add this component and observer on lifecycle
        }

        @OnLifecycleEvent(ON_START) void start() {
                //start listening via system service...
        }

        @OnLifecycleEvent(ON_STOP) void stop() {
                // stop ...
        }

        @OnLifecycleEvent(ON_DESTROY) void cleanup() {
                //remove this observer
        }
}
```

Výpis 2.4: Příklad implementace reakcí na změny stavu pomocí nově představeného mechanismu Lifecycle a LifecycleObserver

Majitel životního cyklu - Lifecycle owner

Jedná se o nové označení prvků uživ. rozhraní, které byly popsány v předchozích kapitolách – Aktivita (kap. 2.2.3) a Fragment (kap. 2.2.4).

2.3.2 LiveData

LiveData je *observable*, který si je vědomo cyklu rozhraní (kap. 2.2.2) – **lifecycle aware**.

Observable – data, která je možné pozorovat – implementovat reakce na změnu těchto dat.

Tato vlastnost má dvě hlavní výhody:

- LiveData vysílají změny pouze těm prvkům uživatelského rozhraní, které jsou v danou chvíli viditelné uživateli (nad majitelem cyklu rozhraní bylo zavoláno **onStart**, případně **onResume**),
- jakmile není nikdo na tyto data přihlášen, všichni "pozorovatelé" jsou automaticky odhlášeni.

LiveData také umožňila zpřehlednit kód. Bez jejich použití při provádění operací na pozadí bylo třeba implementovat rozhraní, které se zavolalo po dokončení, aby se předala hodnota zpět. S použitím LiveDat se stačí přihlásit LiveData, které sami pošlou událost o změně spolu s daty.

2.3.3 Android ViewModel

Android ViewModel je komponenta navržená pro usnadnění a jasné oddělení dat a prvků uživatelského rozhraní.

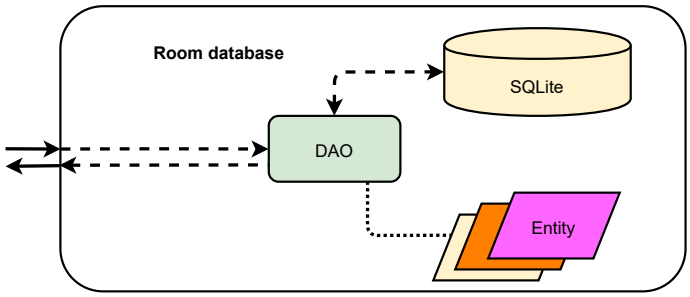
Základním pravidlem pro používání *ViewModel*-u je nemít žádnou referenci na uživ. rozhraní (např. při změně orientace by mohlo dojít k odkazování se na prvek uživ. rozhraní, který už nebude v dané chvíli existovat).

Android ViewModel využívá aplikační context, existuje tedy po celý běh aplikace. Aktivita ani Fragment ViewModel nevytváří – ale spíše žádají systém o jeho přiřazení.

Cílem zavedení této komponenty bylo zbavit aktivitu/fragment jakýchkoliv dat (předejít problémům zmíněným při změně orientace a možnostem úniku paměti). ViewModel není tedy ovlivněn změnami orientace zařízení.

2.3.4 Room databáze

Další komponenta architektury, která má za cíl usnadění vývoj, je **Room** databáze (obr. 2.5). Je konstruována s ohledem na LiveData, pomocí kterých se lze přihlásit na jednotlivé položky v databázi. Mezi její hlavní výhody patří, že korektnost jednotlivých dotazů je validována již při kompilaci. Jedná se o ORM – Object Relational Mapping – napojení záznamů tabulky v databázi na jednotlivé objekty v kódu.



Obrázek 2.5: Room se skládá z SQLite databáze, Entit (abstrakce nad tabulkami) a DAO (SQL příkazy).

SQLite

SQLite je odlehčená verze databáze SQL, není zde žádný vztah klient-server jako u ostatních typů databázi. Data jsou uložena v jediném souboru. Funkcionalita databáze je zakomponována ve zdrojovém kódu.

Entity

S **Room** databází byla představena anotace **@Entity**. Slouží pro anotování objektů, které mají sloužit jako záznamy v databázi.

POJO – akronym pro *Plain Old Java Object*. Jedná se objekt, který má za úkol pouze držet data ve svých proměnných a příslušné funkce pro nastavování (*setter*) a získávání hodnot (*getter*).

Je-li **POJO** anotováno je vytvořena tabulka v databázi korespondující tomuto objektu (výp. 2.5).

```
@Entity(tableName = "schedule_item")
public class ScheduleItem {

        @PrimaryKey(autoGenerate = true)
        private int mUid;

        @ColumnInfo(name = "card")
        private String mCard;

        @ColumnInfo(name = "name")
        private String mName;

        // gettes and setters
}
```

Výpis 2.5: Objekt, který je předlohou pro tabulku v databázi. Jakmile je takovýto objekt anotován pomocí Entity je z něj vytvořena tabulka.

Pomocí anotace zle také nastavit několik parametrů tabulky v databázi. Nejdůležitějším parametrem je název tabulky, jedním z dalších, již nepovinným, parametrem je nastavení primárních klíčů.

Data Access Object

Další nedílnou součástí Room databáze jsou **Data Access Object**-y (dále jen DAO). V kontextu Room databáze se jedná o rozhraní, které definuje příkazy, které se mají provádět nad databází a mají možnost definovat jednotlivé SQL dotazy (výp. 2.6).

Pro operace nad databází jsou k dispozici 4 anotace:

- **@Query** – umožňuje použít přimý SQL dotaz nad databází – jako jediná anotace má parameter – SQL dotaz
- **@Insert**/**@Update**/**@Delete** – vkládání, změny a mazání záznamu – anotuje se jimi funkce, které operují nad záznamem v databázi, který odpovídá Entity objektu v parametru

```
@Dao
public interface ScheduleDao {

        @Query("SELECT * FROM schedule_item
        WHERE mUid LIKE :courseId AND card LIKE :card")
        LiveData<ScheduleItem> get(int courseId, String card);

        @Insert
        void insert(ScheduleItem scheduleItem);

        @Update
        void update(ScheduleItem scheduleItem);

        @Delete
        void delete(ScheduleItem scheduleItem);
}
```

Výpis 2.6: Data Access Object (operace pro přístup) pro záznamy rozvrhu. Pro funkce, které jsou anotovány je vygenerován kód korespondující popisu anotace.

2.4 Material design

Material design lze považovat za sjednocené základní principy designu aplikací (nejen mobilních). Byl představen na Google I/O 2014 a měl za cíl usnadnit vývojářům vytvářet uživatelsky přívětivé aplikace bez potřeby konzultace s profesionalními designery.

Obsahuje velké množství rad a tipů, jak používat jednotlivé prvky uživatelského rozhraní. Další součastí jsou i různé nástroje z nichž jeden je nástroj pro stylizování jednotlivých barev aplikace. Umožňuje navrhnout barevný styl s názornou ukázkou příkladu vzhledu prvků rozhraní. Lze vytvořit barevné schéma pro celou aplikaci na základě jediné vstupní barvy.

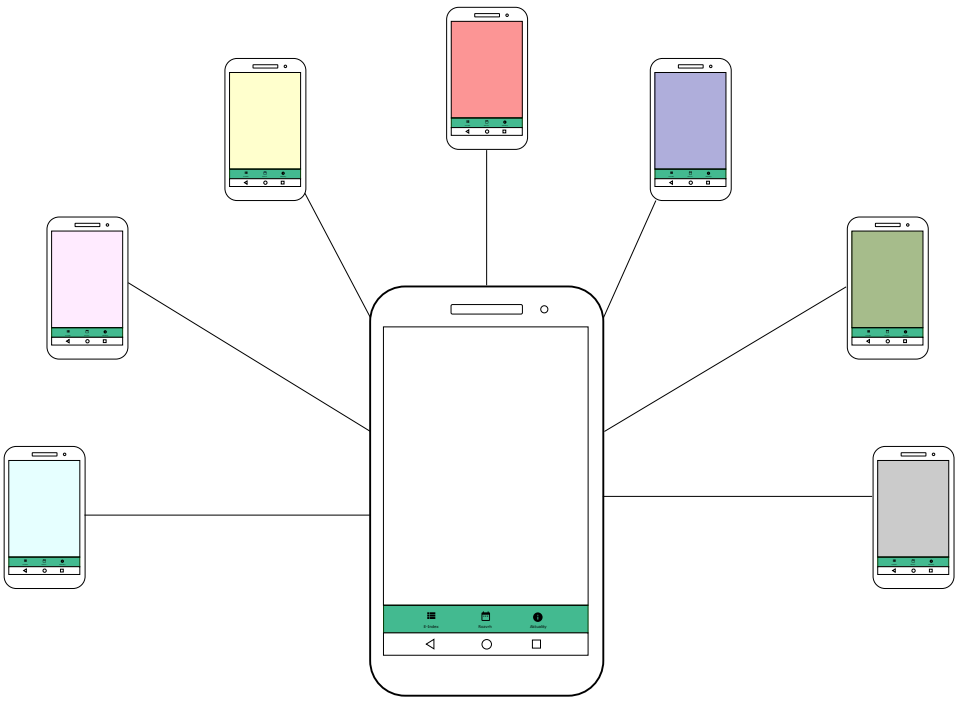
Kapitola 3

Design a architektonický návrh mobilních aplikací

3.1 Navigace v aplikaci

3.1.1 Návrh One Activity Multiple fragments

Princip tohoto návrhu se skládá z jedné hlavní aktivity, která má v sobě navigační prvky aplikace a kontejner, ve kterém se zobrazují fragmenty (obr. 3.1). Toto umožňuje velmi rychlou navigaci skrz aplikaci.



Obrázek 3.1: Aplikace má jednu hlavní aktivitu a několik fragmentů, které se obměňují v obsahovém oknu.

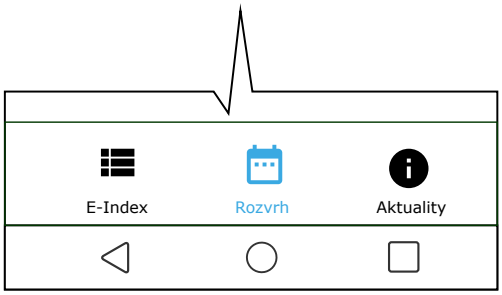
To však neznamená, že aplikace s tímto návrhem by nemohla mít více jak jednu aktivitu. Věci, které nezbytně nesouvisí s hlavním obsahem, jsou obvykle řešeny svými vlastními aktivitami (např. nastavení, informace o aplikaci).

Tento návrh by měl vytvářet abstrakci hlavního obsahu aplikace skrze navigační prvky aktivity – **BottomNavigationBar** a **DrawerLayout**.

Bottom Navigation Bar

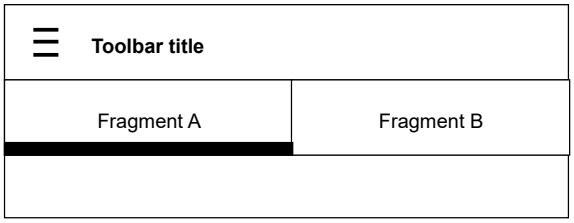
Bottom Navigation Bar je prvek navigace Androidu, který je umístěn naspodu uživatelského rozhraní (obr. 3.2) a měl by obsahovat nejčastěji používané prvky aplikace.

Podle pravidel Material designu[1], by měl avšak mít nejméně 3 a nejvíce 5 prvků. Pro jiné případy existují vhodnější varianty pro navigaci – pro méně **TabLayout** (obr. 3.3) a pro více **DrawerLayout**.



Obrázek 3.2: Navigační prvek, který je zobrazován v hlavní aktivitě, zpravidla pro nejčastěji používané funkce.

Tab Layout

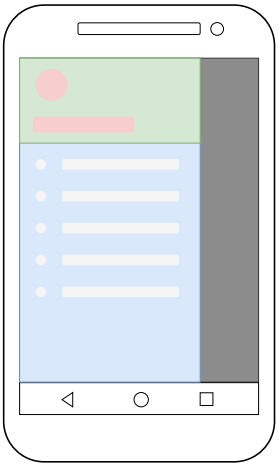


Obrázek 3.3: Vhodný prvek navigace, existují-li právě 2 prvky, mezi kterými chceme přecházet.

Drawer Layout

"Navigační šuplík"[2], dnes již standardem mezi mobilními aplikacemi umožňující navigaci jejich obsahem.

[1]Bottom navigation – https://material.io/guidelines/components/bottom-navigation.html
[2]Drawer – https://material.io/guidelines/patterns/navigation-drawer.html



Obrázek 3.4: Vyjížděcí prvek, který je centrem navigace aplikace

3.1.2 Navigace v rozvrhu

Trend v navigaci aplikací dotykových zařízení je ten, že se uživatel pohybuje v aplikaci spíše pomocí gest, než jednotlivých kliknutí.

Jednou z hlavních funkcionalit aplikace je rozvrh, při jehož ovladání je potenciál použití velkého množství kliknutí (přejít na další den, týden, měsíc), což lze v aplikaci ale nahradit také gesty (obr. 3.5).

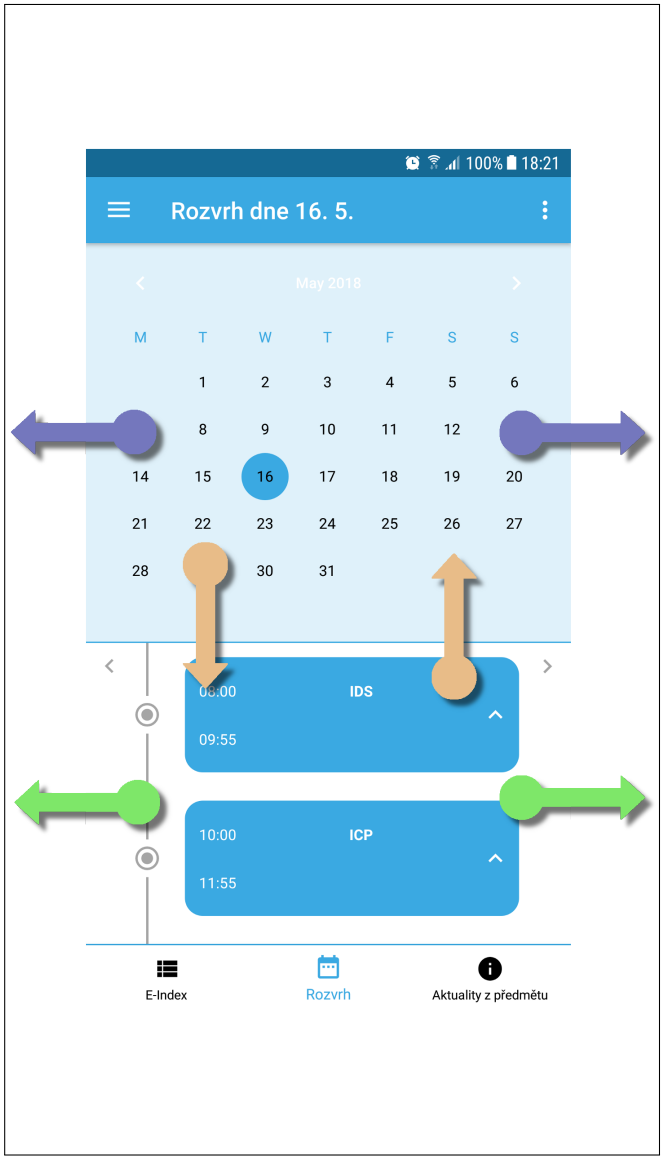
3.2 Návrhový vzor Model View ViewModel

Jedním ze zásadních rozdílů oproti podobným návrhovým vzorům (Model View Controller, Model View Presenter) je ten, že **Model View ViewModel** odhaluje události s daty zároveň spolu s daty jako takovými (obr. 3.6) a vztahy mezi prvky dat a prostředníku mezi uživ. rozhraní. Toto je umožněno pomocí již zmíněných **LiveData** (kap. 2.3.2).

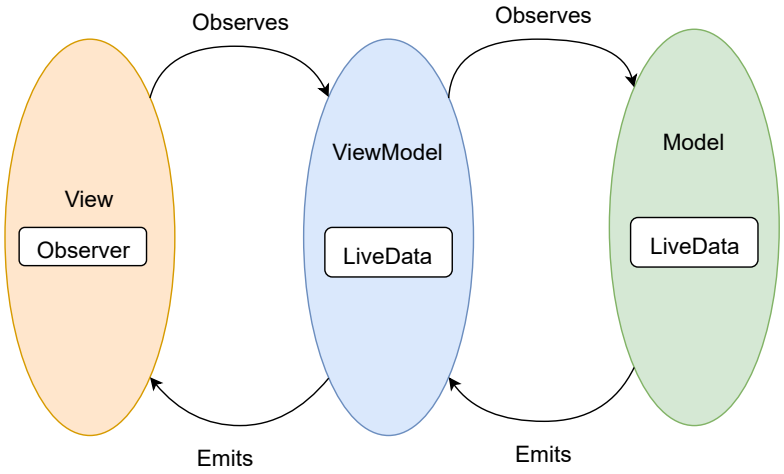
Návrhový vzor je složen, jak už název napovídá, z vrstev:

- **Model** reprezentuje nástroje, data a celkovou *bussiness logiku*[3] aplikace,
- **View** obsahuje elementy uživatelského rozhraní (z pohledu Androidu se jedná jak o rozhraní definované v XML souborech, tak i Aktivity a Fragmenty, které je vytváří),
- **ViewModel** je prostředníkem mezi uživatelským rohzraním a daty.

[3] bussiness logika – získávání dat a operace nad nimi



Obrázek 3.5: Znázornění gest navigace na obrazovce rozvrhu. Gesta (fialová) na prvku kalendáře posouvají měsíce. Vertikální posun (béžová) umožňuje postupné zobrazení pouze rozvrhu pro jednotlivý den. Zelená gesta znázorňují možnost posunu z obsahu rozvrhu po jednotlivých dnech (gesto posunu doleva načte předchozí den a doprava rozvrh pro zítřejší den)



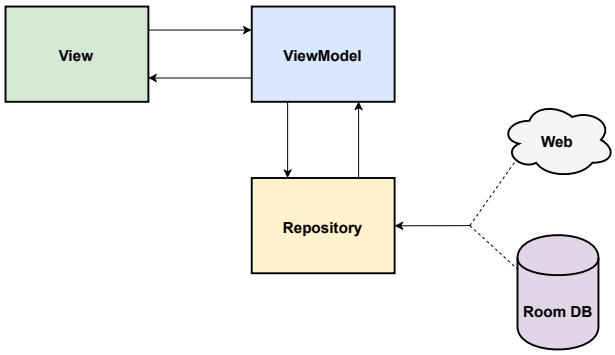
Obrázek 3.6: Propojení jednotlivých vrstev návrhu Model View ViewModel

3.3 Návrhový vzor Repository

Hlavním cílem návrhového vzoru Repository je oddělení dat od uživatelského rozhraní.

Vrstva Repository (obr. 3.7) má na starost:

- Získávání dat skrze síťové připojení.
- Tok dat v proměnných objektů v aplikaci.
- Ukládání získaných dat do místní databáze.
- Plánování a provádění komunikace v pozadí aplikace.



Obrázek 3.7: Znázornění návrhového vzoru Repository. Data oddělena od uživ. rozhraní a centralizovaný přístup k datům. Rozhraní se nemá informace o jejich původu.

3.4 Použité knihovny a technologie

3.4.1 Butterknife

Butterknife[4], je knihovna pro *bindování*[5] objektů uživatelského rozhraní.

Butterknife byl vytvořen kvůli velkému množství kódu, který zhoršoval čitelnost a přehlednost kódu, tzv. *boilerplate kód*[6] (výp. 3.1).

```
Button loginButton = (Button) findViewById(R.id.login_button);
loginButton.setOnClickListener(new OnClickListener() {
        @Override
        public void onClick(View v) {
                // reaction on click
        }
});
```

Výpis 3.1: Řešení přidání reakce na kliknutí tlačítka bez použití knihovny Butterknife.

```
@OnClick(R.id.login_button)
public void onLoginClick(View v) {
        // reaction on click
}
```

Výpis 3.2: Stejná funkcionalita jako 3.1, bez přebytečných řádků kódu.

[4]Butterknife – http://jakewharton.github.io/butterknife/
[5]bindování – spojení elementů definovaných v XML souborech s proměnnými v aktivitách/fragmentech
[6]sekce kódu, který je třeba použít na více místech bez obměn

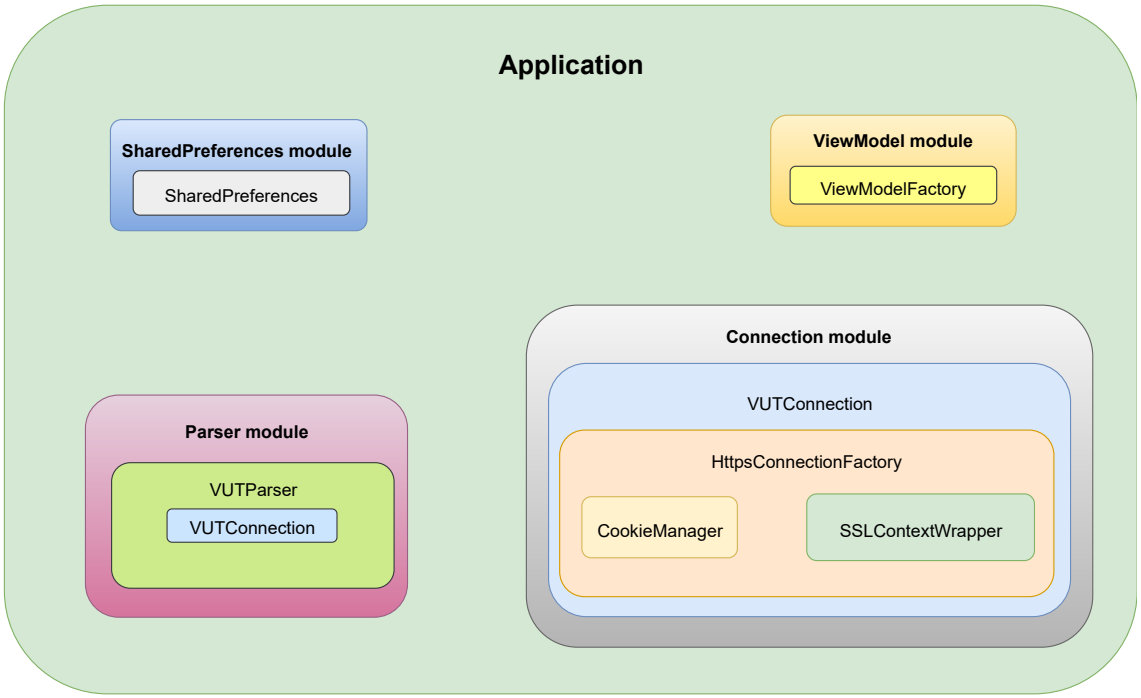
Butterknife funguje na principu anotačního procesoru (výp. 3.2), který je součástí procesu kompilování zdrojových souborů a následného generování kódu (vytvoření potřebných referencí).

Hlavní příčinou tohoto problému bylo to, že *findViewById* (funkce pro získání reference na element uživ. rozhraní v kódu) měla návratovou hodnotu obecný základní View, který musel být následně přetypován na žádaný prvek (výp. 3.1).

Butterknife byl poprvé představen v roce 2013. S příchodem Android 8.0 (Oreo) Google tento problém adresoval a přetypování nutné, nicméně Butterknife má stále výhodu, že je schopen nabindovat více prvků najednou.

3.4.2 Dagger 2

Dagger 2[7] je knihovna od Google-u pro vkládání závislostí (*dependency injection*). Umožňuje modularitu a zjednodušuje implementaci "jedináčků"[8] objektů u kterých je tento přístup potřeba (instance objektu pro práci s databází, uchovávání cookies pro síťové požadavky).



Obrázek 3.8: Graf znázorňující moduly vytvořené pomocí Dagger 2 – je možné získat tyto objekty napříč aplikací

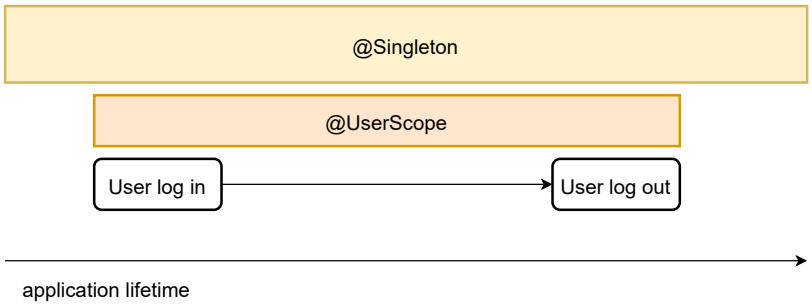
V základu má Dagger 2 pouze jeden rámec (scope) – **@Singleton**. Umožňuje však vytvářet svá vlastní[9].

S možností vytváření vlastních rámců, je možné vytvořit takový rámec, jehož objekty budou mít životnost délky přihlášení uživatele (obr. 3.9).

[7] Dagger 2 – https://google.github.io/dagger/users-guide

[8] singleton – existuje pouze jediná instance dané třídy

[9] Custom scopes – http://frogermcs.github.io/dependency-injection-with-dagger-2-custom-scopes/



Obrázek 3.9: Možnost vytvoření rámce vkládání závislostí s vlastně specifikovanou životností

3.4.3 JSoup

JSoup[10] je open-source knihovna pro procházení a čtení HTML stránek. Umožňuje zaměřování jednotlivých HTML elementu pomocí CSS selectorů.

Jedná se o Java ekvivalent knihovny BeatifulSoup[11] pro Python (oblíbený jazyk pro web scrapping).

3.4.4 Firebase a Crashlytics

Firebase je platforma pro vývoj apliakací, kterou spravuje Google a poskytuje použití i na jiných platformách (iOS, web).

Součást Firebase platformy pro vývoj aplikací je Crashlytics. Crashlytics umožňuje zaznamenávat pády aplikace a podávat hlášení o tom, kde ve zdrojovém kodu pád nastal spolu s anonymními informacemi o zařízení.

Hlášení o pádech aplikace jsou zasílány real-time[12], toto umožnilo během uživatelského testování zachytit chyby a opravovat je během testování.

Součástí platformy je také možnost vzdáleně nastavovat použití AB testů.

Firebase také umožňuje registrovat události v aplikaci. Toto se nejčastěji využívá pro vyhodnocování pohybu po aplikaci a případnou optimalizaci uživatelské zkušenosti.

[10] JSoup – https://jsoup.org/
[11] BeatifulSoup – https://www.crummy.com/software/BeautifulSoup/bs4/doc/
[12] real-time – jakmile nastane pád, je vytvořeno hlášení o chybě

Kapitola 4

Implementace aplikace pro Studis

4.1 Získávání a ukládání dat

Projekt byl započat na základě spolupráce s CVIS[1]. Po konzultacích s vedoucím práce bylo doporučeno odstoupit od záměru získávání dat skrze API a bylo rozhodnuto získávat data skrze webové rozhraní (CVIS nevnímá v současné době projekt jako prioritu a nedostatečná komunikace neumožnila jít cestou vytvoření API). Elektronický index a aktuality z předmětů jsou získávány přes parser (kap. 3.4.3).

Data uživateli jsou uložena do sdílených nastavení a poté se provádí jednotlivé požadavky pro získání dat ohledně předmětů, které jsou ukládány do databáze.

Získávání dat pro rozvrh je řešeno využitím možnosti Studis-u stáhnout rozvrh ve formátu ICS a následné načtení všech událostí do databáze.

Dalším rozdílem oproti načítání předmětů je ten, že rozvrh pro den může být prázdný (podmínka pro kontrolu, zda je třeba načíst předměty z informačního systému). Proto je ve sdíleném nastavení vedena informace o tom, které semestry byly načteny.

Získávání dat rozvrhu je iniciováno, jakmile uživatel přejde na datum, které je mimo již stažené semestry. Aplikace počítá s možnostmi změny rozvrhů. Proto je do nastavení implementována možnost jejich aktualizace.

Stejná možnost je i pro předměty v elektronickém indexu.

4.2 Komunikace

Aplikace implementuje komunikaci dle postupu o bezpečné komunikace od Googlu[2].

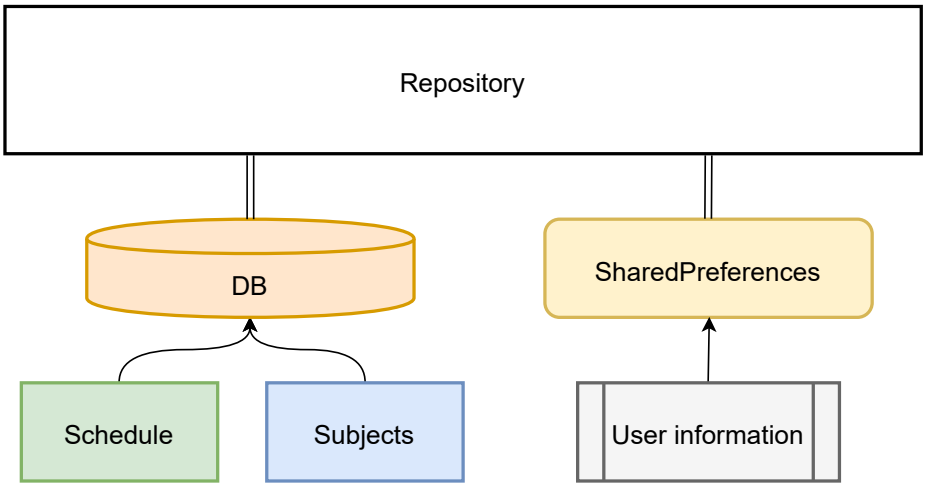
Bezpečnost komunikace prošla hodnocením při nahrání do Google Store bez jakýchkoliv varování. V případě budoucího vývoje se předpokládá změna implementace stylu komunikace, zejména kvůli zvěřejnění zdrojového kódu této aplikace.

4.3 Aktivity

Použitím návrhového vzoru z kap. 3.1.1 jsou v aplikaci použity pouze dvě aktivity, jejichž hlavním úkolem bylo správné zobrazování obsahu skrze fragmenty.

Jejich implementace je však rozdělena do několika tříd zahrnující různé okruhy zoodpovědnosti (obr. 4.2):

[1] CVIS – Centrum výpočetních a informačních služeb
[2] HTTPS a SSL – https://developer.android.com/training/articles/security-ssl



Obrázek 4.1: Demonstrace dat uložených do databáze a do sdílených nastavení.

- **ThemedActivity** – má na starost změnu tématu (barev),
- **ToolbarActivity** – implementuje vytváření toolbaru a poskytuje funkce pro změnu jeho textu
- **FragmentOperationsActivity** – operace s fragmenty
- **NavigationActivity** – rozcestník navigace v aplikaci

4.4 Fragmenty

Fragmenty jsou hlavním prvek uživ. rozhraní této aplikace. Jsou zvoleny zejmnéna díky jejich modularitě.

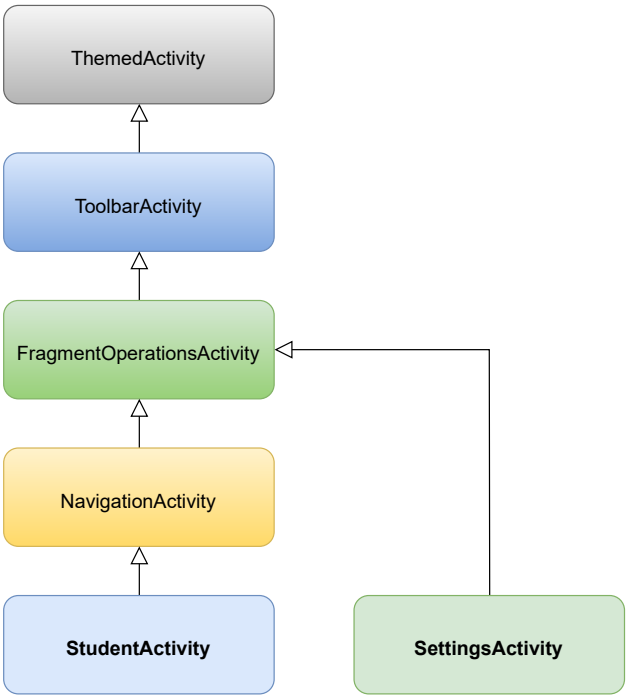
Příkladem zde lze uvést, že seznam předmětů pro semester je řešen samostatným fragmentem, jak pro zimní tak pro letní. Toto umožňuje velmi snadnou adaptací na obrazovky s širším displejem, kde mohou být zobrazeny vedle sebe bez větších změn.

Fragmentů je v aplikaci použito několik (obr. 4.3):

- **TitleFragment** – fragment, který obsahuje implementaci na změnu titulku toolbaru (v hlavička obrazovky). Fragmenty sami o sobě nemají přístup k tomuto elementu. Proto je třeba tyto změny provádět skrze referenci na aktivitu, ve které je fragment umístěn,
- **OptionsFragment** – Options je zde myšlena nabídka v pravém horním rohu
- **SemesterFragment** – obsahuje logiku získávání dat (z ViewModelu) a jejich dosazení do prvku uživ. rozhraní

4.5 Vzhled aplikace

Jelikož jednotlivé fakulty VUT jsou barevně navzájem odlišeny, rozhodl jsem se to zoohlednit i při návrhu a vytvářel jsem uživ. rozhraní s myšlenkou, že se aplikace po příhlášení studenta přizpůsobí na barvě jeho fakulty.



Obrázek 4.2: Hierarchie dědičnosti tříd aktivit.

Toto je v aplikaci umožněno pomocí **témat** (*themes*). Téma je soubor atributů, které se používají interně při vytváření prvků rozhraní (výp. 4.1).

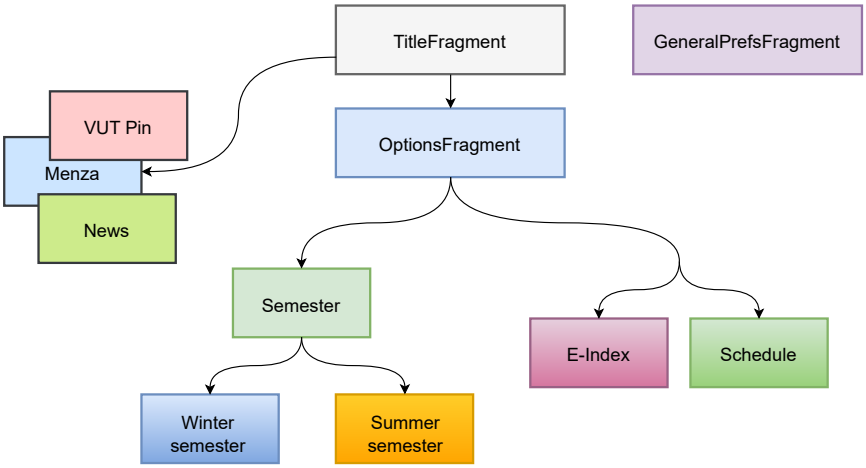
```
<style name="MerkurAppTheme.FIT" parent="MerkurAppTheme">
        <item name="colorPrimary">@color/fit_colorPrimary</item>
        <item name="colorPrimaryDark">@color/fit_colorPrimaryDark</item>
        <item name="colorAccent">@color/fit_colorPrimary</item>
        <item name="android:textColor">@android:color/white</item>
        <item name="colorButtonNormal">@color/fit_colorPrimary</item>
</style>
```

Výpis 4.1: Příklad deklarace tématu.

Téma je nastavováno jednotlivé aktivitě v manifestu. Android jako takový sám neposkytuje možnost dynamické změny tématu.

Zmíněné chování bylo docíleno pomocí SharedPreferences (kap. 2.2.6), ve kterých je uložena informace o aktuální barvě/fakultě. Jakmile je zde tato informace uložena všechny aktivity v aplikaci jsou znovu vytvořený s respektem vůči této změně. Tato logika je implementována v **ThemedActivity** (obr. 4.2), která je předkem všech aktivity, které jsou v aplikaci použity a tudíž je toto chování zaručeno po celé aplikaci.

Aby tato změna fungovala je třeba nastavit téma, předtím než je jakýkoliv element uživ. rozhraní vytvořen (tj. nastavení tématu by měla být první věc v **onCreate**).



Obrázek 4.3: Hierarchie dědičnosti tříd fragmentu.

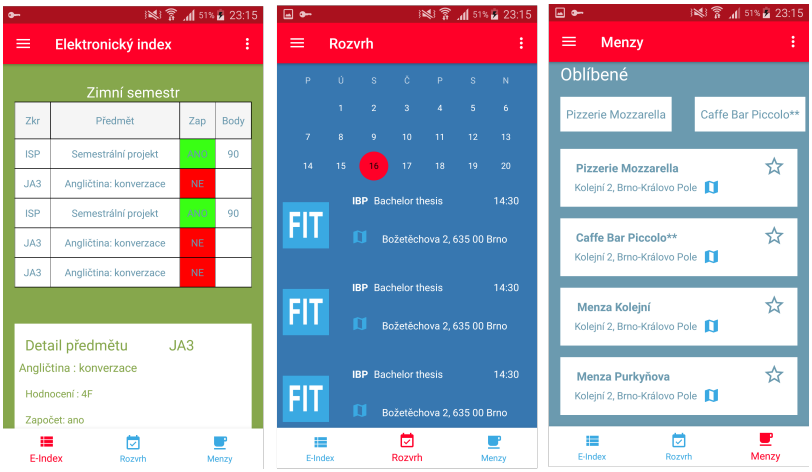
Kapitola 5

Testování a zpětná vazba

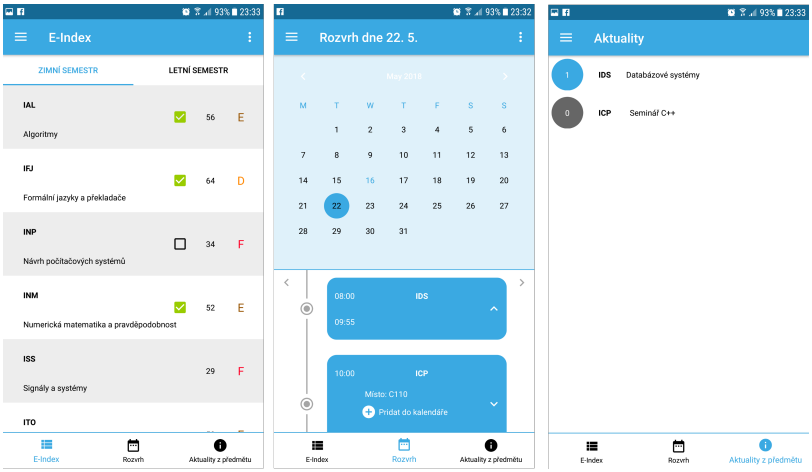
Vytvořená aplikace byla ve spolupráci se Studentskou unií FIT VUT testována studenty.

Testování se zúčastnilo 15 studentů. Testování aplikace studenty bylo na pokyn z CVIS bohužel přerušeno, protože chtějí studenty o extistenci aplikace informovat později a vlastní cestou tak, aby bylo zajištěno splnění Obecného nařízení o ochraně osobních údajů (GDPR) a aby byla promyšlena PR strategie představení aplikace (v minulosti se stalo, že informaci o aplikaci vzniklé v rámci studentské práce převzala média před oficiálním oznámením VUT). V budoucnu bude tedy potřeba v testování aplikace studenty pokračovat, s oficiální záštitou VUT.

Studentům byly poskytnuty 2 prototypy aplikace k instalaci a testování (prvotní (obr. 5.1) a pokročilejší prototyp). K testování byl připojen krátký dotazník, který zkoumal zájem o mobilní aplikaci pro Studis VUT, zda se jim aplikaci podařilo nainstalovat, zda-li je ovládání aplikace intuitivní atp. Ve finální verzi byli zohledněny a přídány některé prvky/funkcionality, které studenty napadli (obr. 5.2).



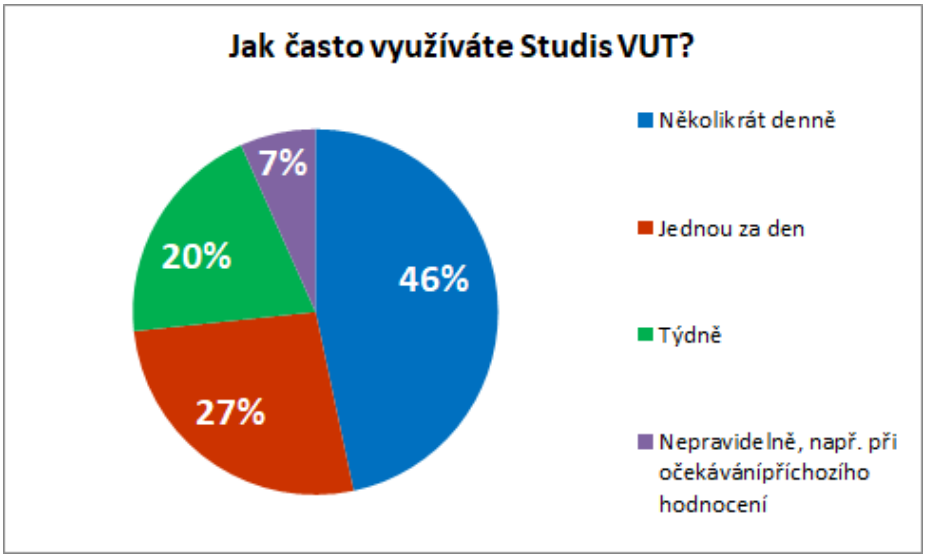
Obrázek 5.1: Letmý pohled na první prototyp vytvořený během prvních dvou týdnu po diskuzi o možnosti tohoto projektu.



Obrázek 5.2: Náhled finální verze aplikace.

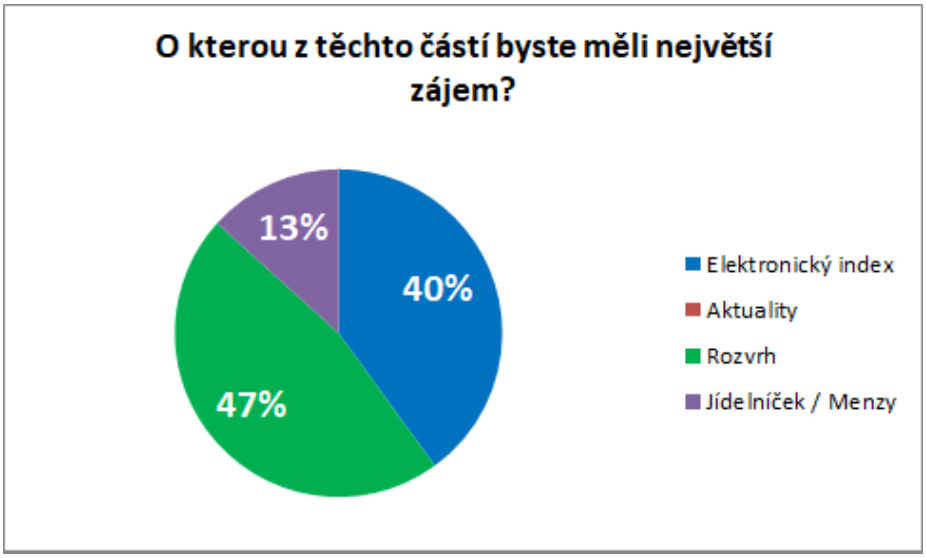
5.1 Výstup od studentů

Všichni dotazovaní studenti by dle dotazníku aplikaci pro Studis VUT ocenili, většina studentů informace z něj využívá několikrát denně – obr. 5.3.



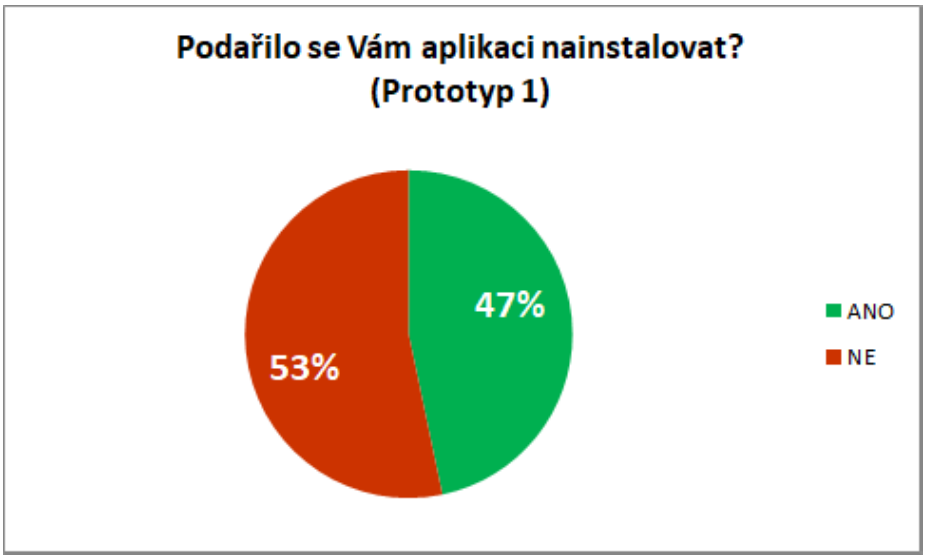
Obrázek 5.3: Četnost navštěvování Studisu.

Největší zájem studenti mají o rozvrh a elektronický index – obr. 5.4.

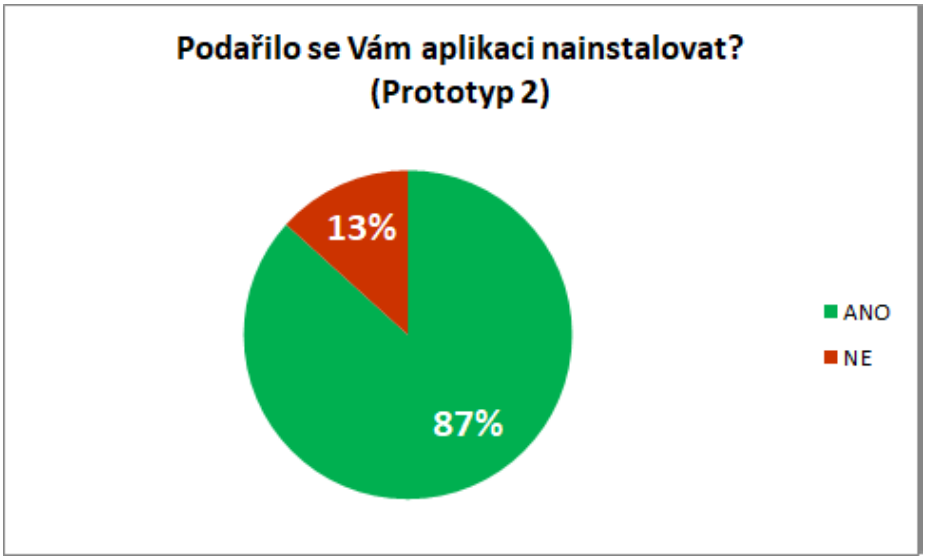


Obrázek 5.4: Zobrazení zájmu o jednotlivé prvky Studis-u/aplikace.

U prvního prototypu měli studenti problémy s instalací (53 % procentům se instalace aplikace nepodařila – problém při generování instalačích souborů) – obr. 5.5. Pokročilejší prototyp měl při instalaci úspěšnost vyšší (87 %) – obr. 5.6. V komentářích od studentů se objevily poznámky o pádu aplikace, například u zařízení Xiaomi Note 4 Global, je tedy třeba se na tento problém ještě více zaměřit.

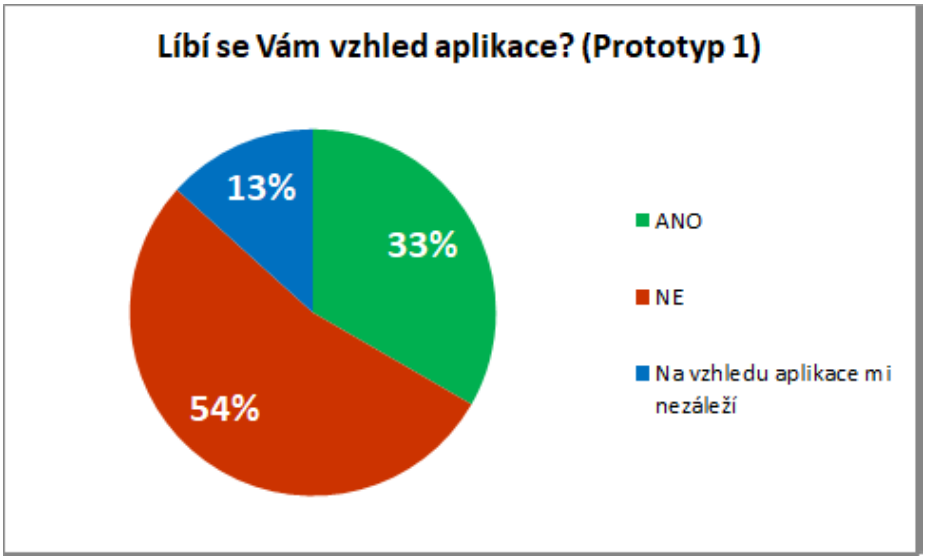


Obrázek 5.5: Úspěšnost instalace prvního prototypu.

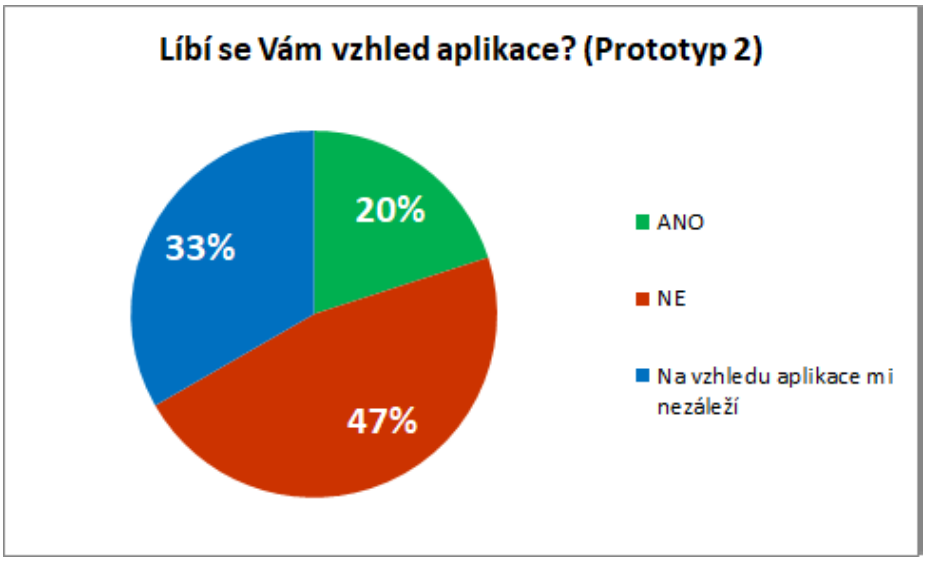


Obrázek 5.6: Úspěšnost instalace druhého prototypu.

Co se týče vzhledu aplikace, je vidět jisté zlepšení v hodnocení druhého prototypu oproti prvnímu (obr. 5.7 a obr. 5.8), ale stále je potřeba na vzhledu aplikace pracovat. Hodnocení, že se aplikace studentům nelíbí, volili také studenti, kterým se aplikaci napodařilo nainstalovat, což tedy hodnocení vzhledu aplikace zkresluje. Tento problém byl vyřešen nahráním a zveřejněním beta verzi na Google Play store (obr. B.1)

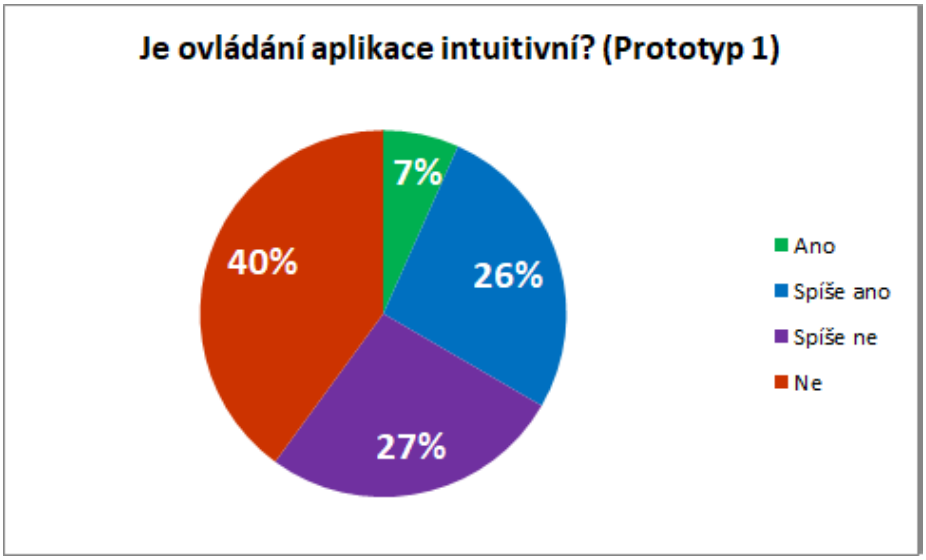


Obrázek 5.7: Hodnocení vzhledu prvního prototypu.

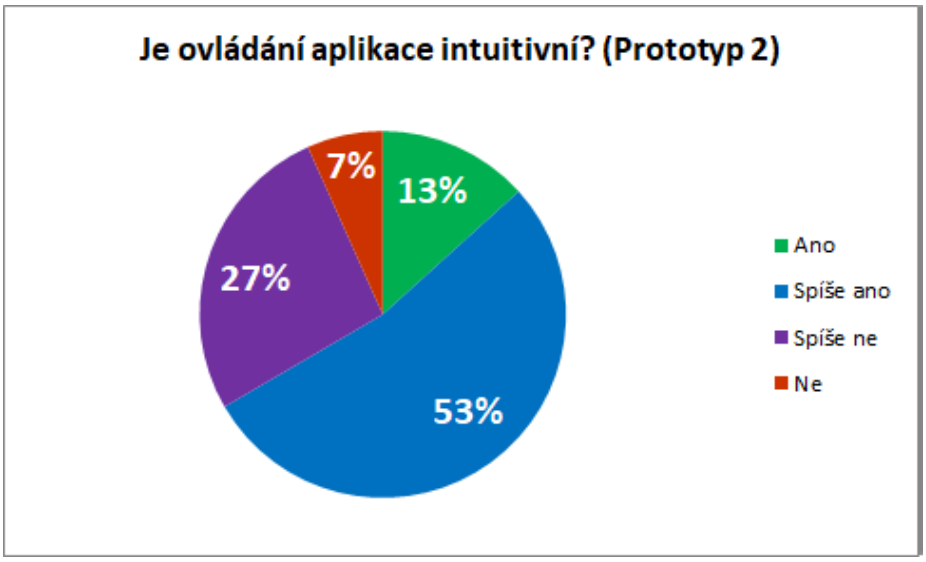


Obrázek 5.8: Hodnocení vzhledu druhého prototypu.

Z odpovědí na otázku týkající se ovládání aplikace vyplývá, že ovládání druhého prototypu (obr. 5.10 oproti obr. 5.9) je výrazně intuitivnější, opět je to však může být zkresleno negativními odpověďmi v případě neúspěšného instalování aplikace.



Obrázek 5.9: Intuitivita rozhraní pro uživatele v prvním prototypu.



Obrázek 5.10: Hodnocení intuitivnosti ovládání aplikace u druhého prototypu.

Kapitola 6

Závěr

Bakalářská práce Mobilní aplikace pro Studis VUT se zabývá vývojem mobilní aplikace pro informační systém VUT v operačním systému Android.

V teoretické části práce jsou popsány základní elementy při tvorbě aplikace v operačním systému Android, jako je například Android manifest, základní prvky komunikace v systému Android - intenty a jejich filtry, životní cyklus prvků uživatelského rozhraní (prvky aktivita a fragment) a komponenty architektury Androidu. Práce také rozebírá design a architektonický návrh mobilních aplikací a zaměřuje se na návrhový vzor Model View ViewModel.

Implementace aplikace se zaměřila na studenty nejpoužívanější komponenty informačního systému, tedy na rozvrh, index a aktuality z předmětu. Hlavním úspěchem aplikace je autentizace přihlašování přes VUT login a heslo, po přihlášení tedy se tedy studentovi zobrazují reálné aktuální informace z informačního systému VUT. Pro anonymizaci (GDPR) testování rozhraní byli vygenerovány data, ke kterým se dá dostat bez vkládání osobních údajů. Na přihlášení studenta reaguje vzhled aplikace, který se přizpůsobí barvě fakulty, ze které student pochází. Podařila se také implementace rozvrhu, který zobrazuje aktuální rozvrh přednášek daného studenta a pomocí lišty lze procházet jednotlivé události v daném dni.

Vytvořená aplikace byla testována na studentech ve spolupráci se Studentskou unií FIT VUT, do testování se zapojilo 15 studentů. Testování aplikace studenty bylo na pokyn z CVIS bohužel přerušeno, protože chtějí studenty o extistenci aplikace informovat později a vlastní cestou tak, aby bylo zajištěno splnění Obecného nařízení o ochraně osobních údajů (GDPR) a aby byla promyšlena PR strategie představení aplikace. V budoucnu bude tedy potřeba v testování aplikace studenty pokračovat, pod oficiální záštitou VUT.

Studentům byly poskytnuty 2 prototypy aplikace k instalaci a testování (prvotní a pokročilejší prototyp). K testování byl připojen krátký dotazník, který zkoumal zájem o mobilní aplikaci pro Studis VUT, zda se aplikaci podařilo nainstalovat, zda-li je ovládání aplikace intuitivní atp. Z dotazníku vyplývá, že o mobilní aplikaci pro Studis VUT mají studenti velký zájem, což vedle odpovědi v dotazníku zdůrazňovali i na konci dotazníku v prostoru pro komentáře. Co se týče srovnání prvního a druhého prototypu, došlo k výraznému zlepšení úspěšnosti instalace aplikace (ze 47 % na 87 %) a ke zvýšení intuitivnosti ovládání aplikace. Naopak vidím výrazný prostor pro lepší propracování designu aplikace.

Pro další testování aplikace na studentech bych lépe volil otázky pro dotazník, zaměřil bych se více na otázky otevřeného typu (případně doplňující otázky k otázkám uzavřeným) a přidal bych možnost přidání například printscreenu chyb, na které uživatelé při používání narazí.

Zpracování této aplikace bude ve spolupráci s CVIS pravděpodobně dále vyvíjena. Dalšími prvky, které by mohly být do aplikace přidány, jsou: možnost nastavení oznámení na obrazovku v případě přidání nového hodnocení v indexu či upozornění na volný termín zkoušky, možnost psaní poznámek nebo upomínek, přihlašování na zkoušky.

Součástí práce je návrh plakátu a krátké video pro prezentování aplikace, které jsou přiloženy v příloze této práce.

Literatura

[1] Alabaster, J.: *Android founder: We aimed to make a camera OS*. PCWorld, Duben 2016, [Online; navštíveno 05.04.2018].
URL https://www.pcworld.com/article/2034723/android-founder-we-aimed-to-make-a-camera-os.html

[2] CommonsWare, L.: *Android's Architecture Components*. [Online; navštíveno 16.05.2018].
URL https://commonsware.com/AndroidArch/

[3] CommonsWare, L.: *The Busy Coder's Guide to Android Development*. [Online; navštíveno 16.05.2018].
URL https://commonsware.com/Android/

Příloha A

Obsah přiloženého paměťového média

- **merkur_thesis** – kompletní projekt z Android studia (bez generovaných souborů)
- **latex** – technická zpráva ve formě zdrojového kódu LaTeX-u
- **xsmyck01_IBP.pdf** – technická zpráva této práce
- **xsmyck01_IBP.mp4** – video zobrazující průchod aplikací
- **plakat1.pdf, plakat2.pdf** – plakátky pro aplikaci

Příloha B

QR kód na Google Play store

Obrázek B.1: Odkaz na betu verzi v Google Play